Pacific Northwest
NATIONAL LABORATORY

MCL + Alternative Resources

PPoPP ‘21
Feb 27 2021

Ryan Friese, Roberto Gioiosa, Alok Kamatar

U.S. DEPARTMENT OF ENERGY BATTELLE

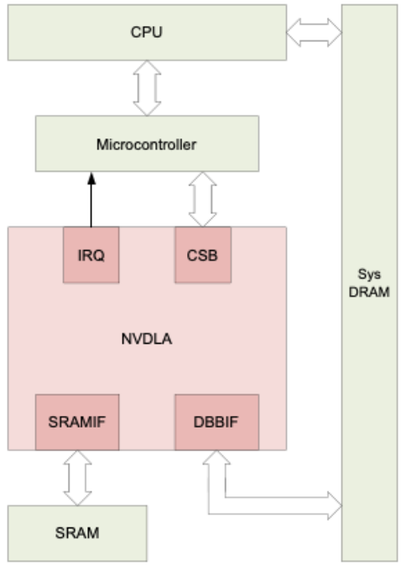
PNNL is operated by Battelle for the U.S. Department of Energy

Tutorial Outline

- Some prep work first
  - NVDLA background
  - Actual hardware + Emulation
- Then the fun stuff – performing inference on NVDLA emulators
  - pulling the mcl ppopp21 tutorial docker (docker pull minoscomputing/ppopp21)
    - ✓ If you haven't done so already go ahead a download now
  - Stepping through an example application
  - Launching NVDLA emulated devices
  - Launching MCL scheduler
  - Executing our test app!

NVDLA – Nvidia Deep Learning Accelerator

- Accelerates compute effort of deep learning inference
  - 4 main groups
    - ✓ Convolutions
    - ✓ Activations
    - ✓ Pooling
    - ✓ Normalization
  - Share a few similar Characteristics
    - ✓ Regular/Predictable memory access
    - ✓ Readily parallelizable
- Goal: standardized, open architecture for deep learning inference
  - Scalable
  - Configurable (w.r.t. overall accelerator)



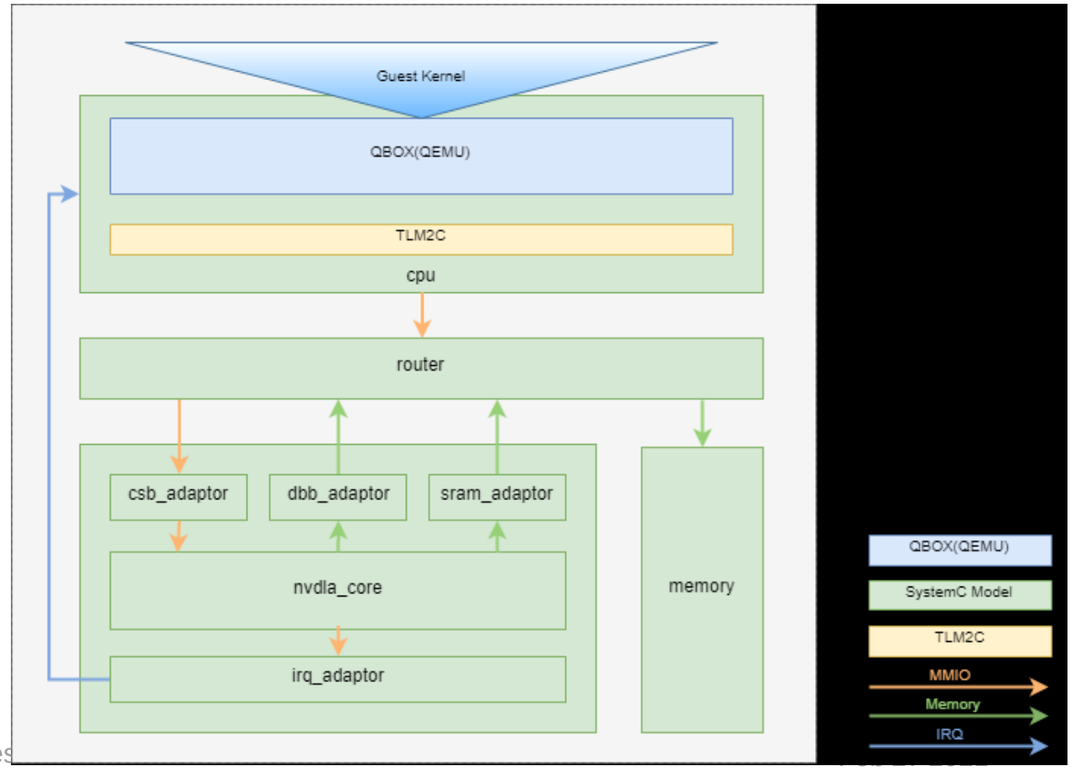
Nvidia Jetson AGX Xavier

- Nvidia developed DLA
- Headed implementation
  - Controller coprocessor closed source

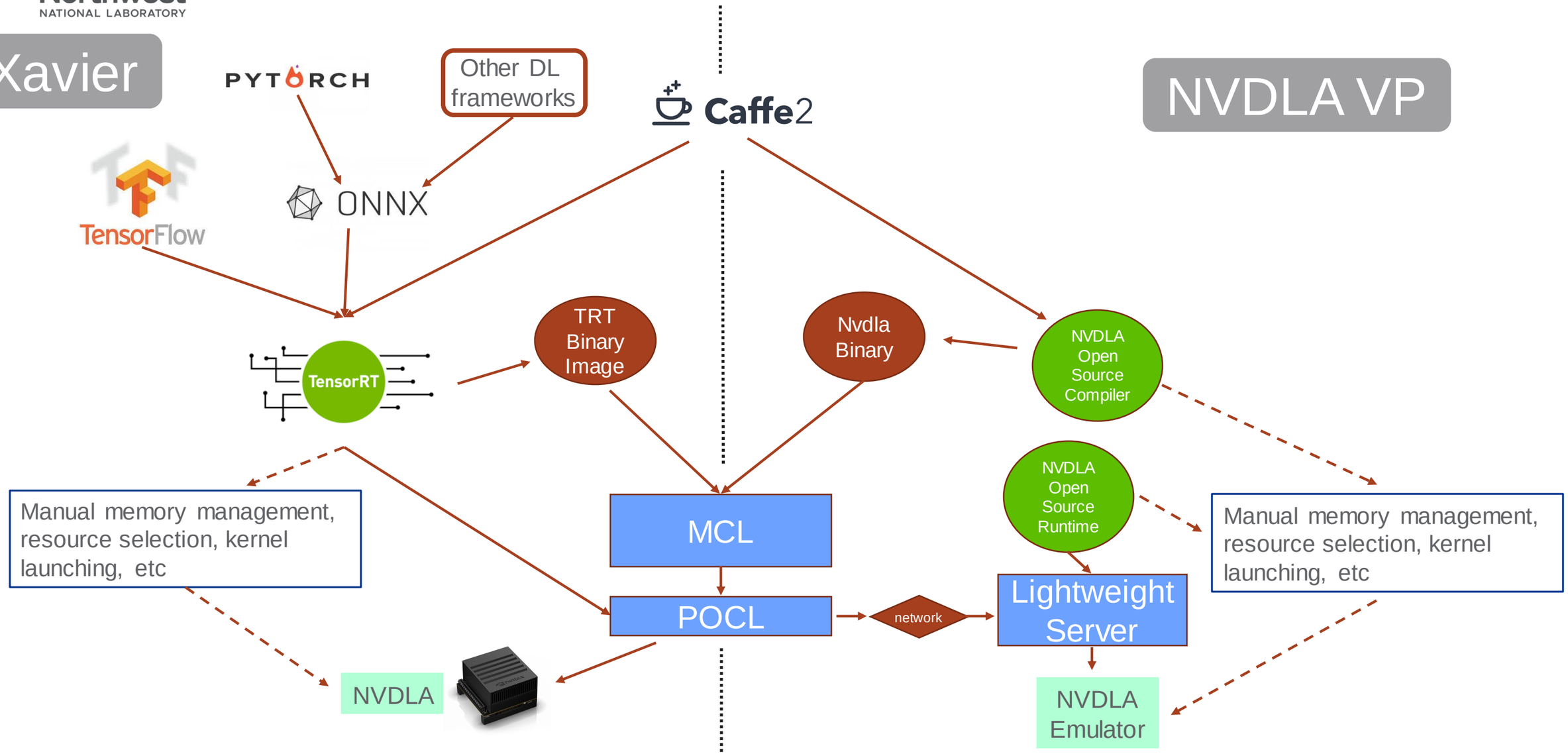
GPU	512-core Volta GPU with Tensor Cores
CPU	8-core ARM v8.2 64-bit CPU, 8MB L2 + 4MB L3
Memory	32GB 256-Bit LPDDR4x | 137GB/s
Storage	32GB eMMC 5.1
DL Accelerator	(2x) NVDLA Engines

Nvidia NVDLA Virtual Platform

- The Open Source offering provided by Nvidia (http://nvdla.org/vp.html)
  - QBOX: System-C + Qemu based emulation (what we will be using)
  - Register accurate emulation
  - (although not Virtual, the same open source hardware + software works with FPGAs)
  - From our point of view we interact with the emulated devices as if they are network attached accelerators
    - ✓ We run a lightweight server process on the guest server
  - You could conceivably run a full-fledged OS on the guest and interact with as a local accelerator

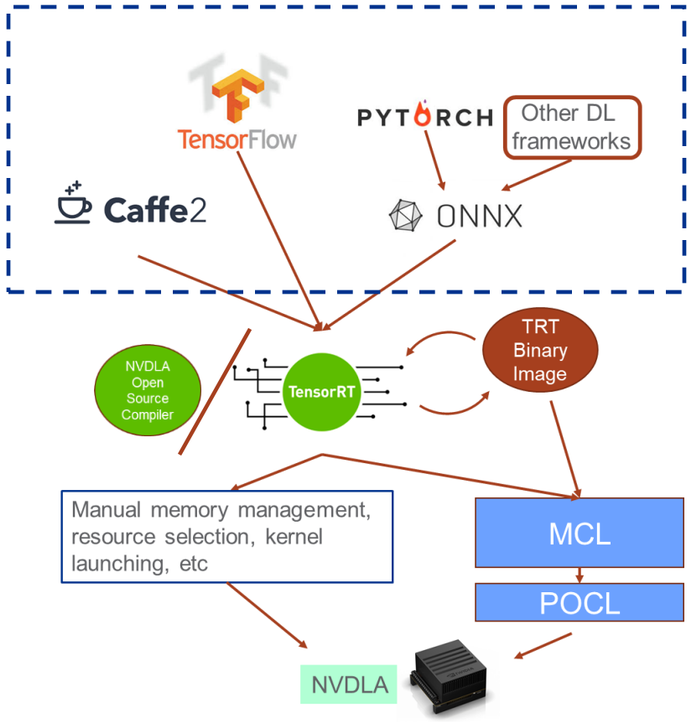


DLA Stack


Xavier
NVDLA VP
PYTORCH
Other DL frameworks
Caffe2
TensorFlow
ONNX
TensorRT
TRT Binary Image
Nvdla Binary
NVDLA Open Source Compiler
Manual memory management, resource selection, kernel launching, etc
MCL
POCL
network
NVDLA Open Source Runtime
Lightweight Server
Manual memory management, resource selection, kernel launching, etc
NVDLA
NVDLA Emulator

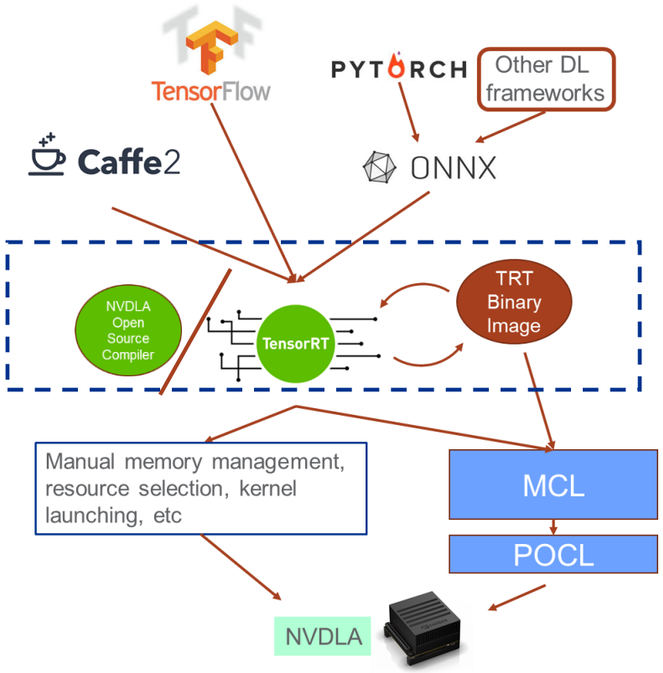
Model Construction and Training

- Models implemented using high level ML API's
- Training performed offline
  - Potentially on more powerful systems
- Networks + Weights serialized
  - ONNX
  - Caffe
  - UFF



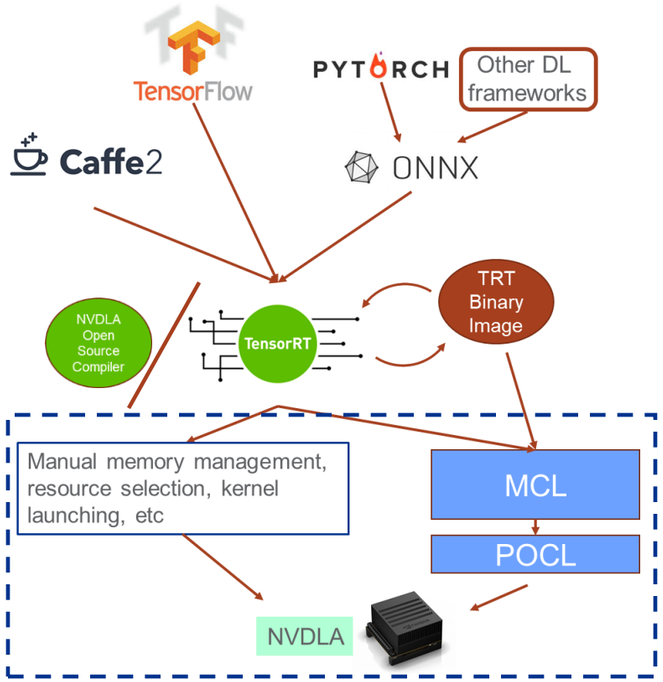
Inference Engine Creation

- Ingest network definition & weights
  - Or define network manually using TensorRT/OpenSource Runtime
- Specify parameters (some dependent on network definition)
  - Devices (Which DLA Core, allow GPU fallback)
  - Batch size
  - FP mode
- Serialize inference engine
  - NVDLA binary executable
- On the Xavier use `trtexec`
- On the virtual platform use `nvdla_compiler`



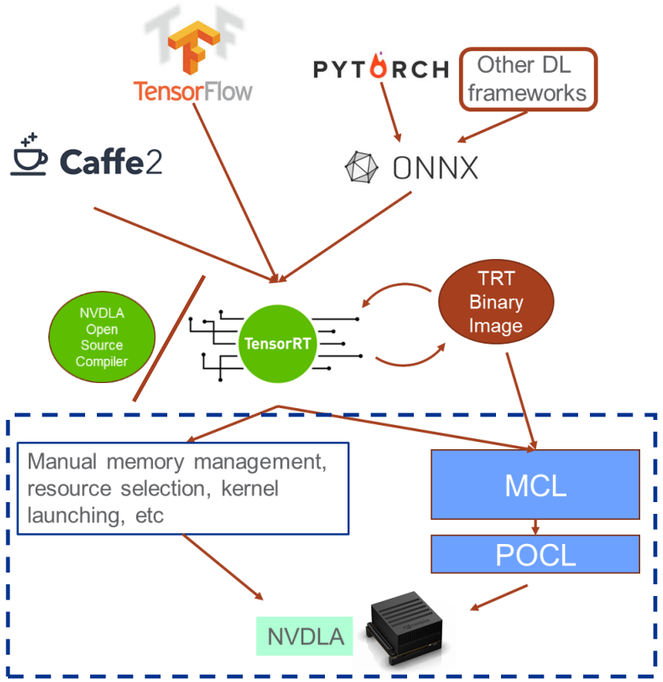
Model Execution (Inference) on Xavier

- Deserialize binary into InferenceEngine
- Construct ExecutionContext
- Allocate and load input/output buffers
  - Unified memory on the Xavier
- Create and initialize cudaStream (work queue)
- Enqueue inference task
- Wait on task completion
- Cleanup
- All implemented using the TensorRT C++ API
  - Also requires some additional bookkeeping and ancillary data structures



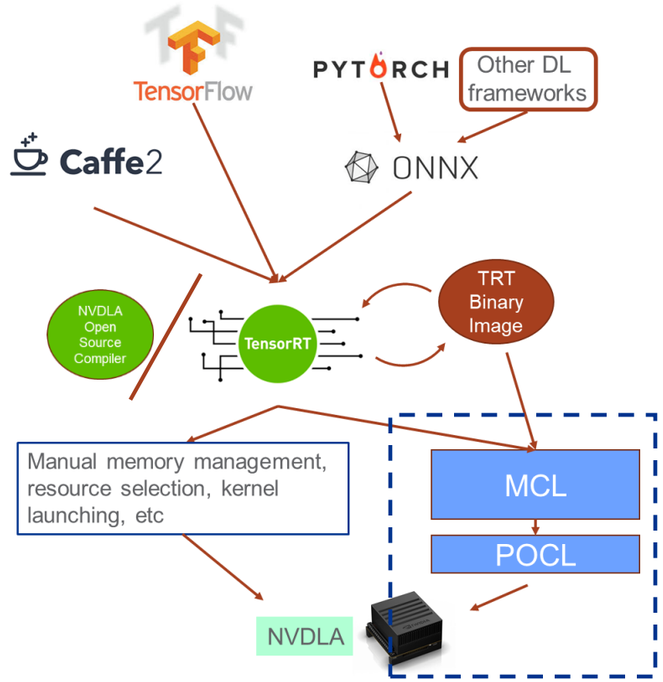
Model Execution (Inference) on Virtual Platform

- Create Nvdla Runtime
- Load model & Initiate EMU
  - This is roughly the equivalent of a ExecutionContext
  - Only one model can be loaded at a time (per NVDLA “Core”)
- Allocate and load input/output buffers
  - A call to allocate a separate call to bind to current model
- Submit inference task
  - Synchronous operation
- cleanup
- All implemented using the NVDLA OpenSource C++ API
  - There are multiple layers of API available to interact with the devices, we currently are using the highest



MCL – NVDLA integration

- Recall: MCL is built on top of OpenCL
- Recall: Tasks are OpenCL kernels (source code) and associated inputs/outputs
  - Compiled/executed depending on the device a task runs on
  - Devices managed by the MCL Scheduler
- NVDLA does not have an OpenCL implementation nor does it compile directly from source
- For integration we have developed a custom POCL[1] device for the NVDLA
- Currently, ingests preconstructed DLA binary
- Looking into direct construction from an ML deployment model (e.g. ONNX)
  - Would eliminate user needing to run “trtexec” or “nvdla_compiler” or implement a custom serializer



[1] http://portablecl.org/

NVDLA POCL Driver

- POCL – Portable Compute Language
  - Open source implementation of OpenCL standard
- Provides the connection between MCL and NVDLA
  - Xaiver – calls directly into TensorRT
  - Virtual Platform – connects to our lightweight server running in the Qemu image
- Device discovery and initialization
- Buffer/memory management
- Launches and reports finished execution of tasks
- Implemented using the OpenCL “builtin_kernel” interface
  - Specifies that the device doesn’t run arbitrary OpenCL code

Pacific Northwest NATIONAL LABORATORY

Time to test it out!

- Hopefully by now your docker image has finished downloading
  - Recall: “docker pull minoscomputing/ppopp21”
- Goal: Take a pretrained mnist (digit recognition) model and use it to performance inference on an (emulated) NVDLA
- We will perform the following steps:
  - Instantiate NVDLA Qemu devices
  - Compile mnist caffe model into an nvdla binary
  - Launch MCL scheduler process with POCL envargs to enable NVDLA’s
  - Perform inference using an example MCL application

Instantiating NVDLA Qemu Devices – Host Side

```
host:~# docker run –name ppopp21 –it minoscomputing/ppopp21 /bin/bash
```

```
root@container:/ppopp21# cd nvdla/
```

```
root@container:/ppopp21/nvdla#./start_nvdla_emulator.sh 2
```

We can create multiple emulated NVDLAS!

Each instance forwards a unique host port

NVDLA Qemu configuration

```
#!/bin/bash
export SC_SIGNAL_WRITE_CHECK=DISABLE
NVDLA_EMU_BASE=/usr/local/nvdla
TUTORIAL_DIR=$PWD

num_instances=$1
if [ -z "$num_instances" ]; then
    num_instances=1
fi

mkdir -p nvdla_emulator_configs
base_port=6000

for i in `seq 0 $(( num_instances-1 ))`; do
port=$(( base_port + i ))
```

```
cat <<EOT > nvdla_emulator_configs/port${port}.lua
CPU = {
    library = "libqbox-nvdla.so",
    extra_arguments = "-machine virt -cpu cortex-a57 -machine type=virt \
    -nographic -smp 1 -m 1024 -kernel Image --append \"root=/dev/vda\" \
    -drive file=rootfs_${port}.ext4,if=none,format=raw,id=hd0 \
    -device virtio-blk-device,drive=hd0 -fsdev local,id=r,path=.,security_model=none \
    -device virtio-9p-device,fsdev=r,mount_tag=r -netdev user,id=user0,hostfwd=tcp:${port}-:6667, \
    -device virtio-net-device,netdev=user0"
}

ram = {
    size = 1048576,
    target_port = {
        base_addr = 0xc0000000,
        high_addr = 0xffffffff
    }
}

nvdla = {
    irq_number = 176,
    csb_port = {
        base_addr = 0x10200000,
        high_addr = 0x1021ffff
    }
}
EOT
```

```
cd $NVDLA_EMU_BASE
cp rootfs.ext4 rootfs_${port}.ext4
expect ${TUTORIAL_DIR}/run_nvdla_emulator.exp \
${TUTORIAL_DIR}/nvdla_emulator_configs/port${port}.lua &
cd ${TUTORIAL_DIR}
done
```

Each instance creates a copy of the virtual hard drive

Instantiating NVDLA Qemu Devices – Guest side

```
host:~# docker run –name ppopp21 –it minoscomputing/ppopp21 /bin/bash
```

```
root@container:/ppopp21# cd nvdla/
```

```
root@container:/ppopp21/nvdla#./start_nvdla_emulator.sh 2
```

"expect" is a program that allows us to talk to other interactive application via a script

```
43  cd $NVDLA_EMU_BASE
44  cp rootfs.ext4 rootfs_${port}.ext4
45  expect ${TUTORIAL_DIR}/run_nvdla_emulator.exp \
46  ${TUTORIAL_DIR}/nvdla_emulator_configs/port${port}.lua &
47  cd ${TUTORIAL_DIR}
48  done
```

```
#!/usr/bin/expect -f
set timeout -1
log_file my_log_file.log

spawn aarch64_toplevel -c [lindex $argv 0]

expect "nvdla login: "
send "root\n"
expect "Password: "
send "nvdla\n"

expect "# "
send "mount -t 9p -o trans=virtio r /mnt\n"
expect "# "
send "cd /mnt\n"
expect "# "
send "insmod drm.ko\n"
expect "# "
send "insmod opendla_1.ko\n"

expect "# "
send "LD_LIBRARY_PATH=. ./nvdla_emu_server\n"
expect "# "
```

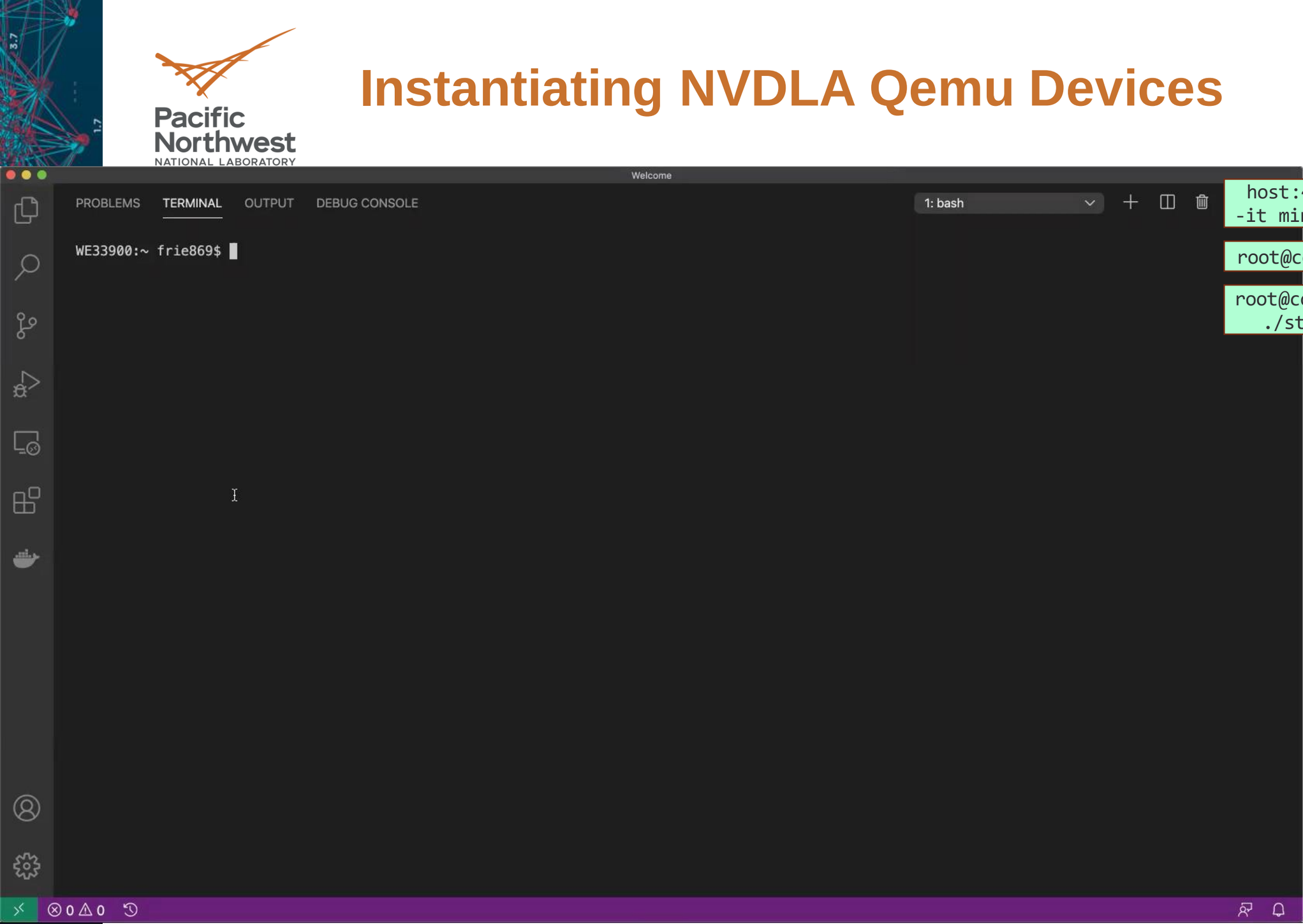
Launch the Qemu instance

Mount from the host the directory where this image was launched

Insert the NVDLA device driver module into the driver

Launch lightweight server

Instantiating NVDLA Qemu Devices



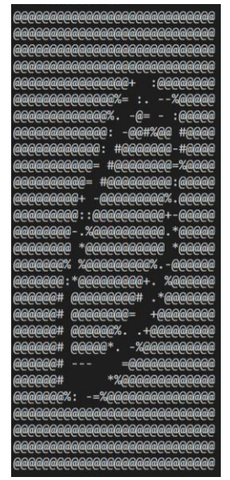

```
host:~# docker run -name ppopp21 \
-it minoscomputing/ppopp21 /bin/bash
```

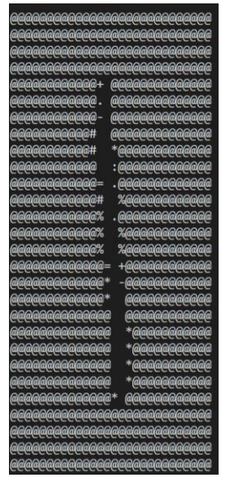
```
root@container:/ppopp21# cd nvdla/
```

```
root@container:/ppopp21/nvdla# \
  ./start_nvdla_emulator.sh 2
```

Compiling MNIST model to NVDLA format

```
# we want our qemu images running so open a new terminal
host:~/#docker exec -it ppopp21 /bin/bash
root@container:/ppopp21# cd nvdla/
root@container:/ppopp21/nvdla# make caffe=mnist/mnist.caffemodel \
            prototxt=mnist/mnist.prototxt nvdla_emu_bin
```



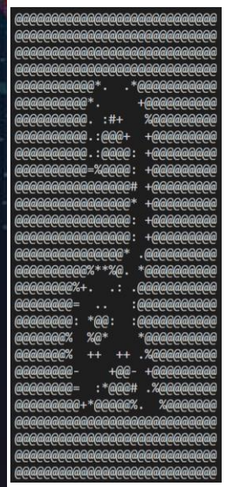


```
NVDLA_EMU_BASE=/usr/local/nvdla
NVDLA_EMU_COMPILER=$(NVDLA_EMU_BASE)/nvdla_compiler
LD_LIBRARY_PATH=$(NVDLA_EMU_BASE) $(NVDLA_EMU_COMPILER) --prototxt ${prototxt} --caffemodel ${caffe}
mv fast-math.nvdla mnist/mnist.nvdla
rm output.protobuf
rm -r wisdom.dir
```

Nvidia Open Source nvdla compiler (comes prebuilt on the virtual platform)

Rename from default

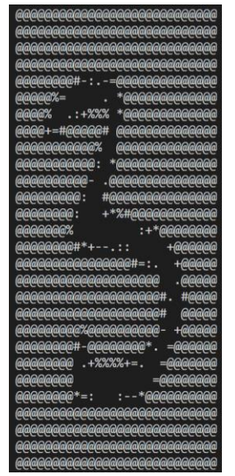
Rename from default



The process on the Xavier is very similar but uses a different compiler

In theory NVDLA binaries should be compatible across different implementations assuming the layers with the network are present in the accelerator

We have not (yet) been able to use the same binary on the Xavier and the emulated NVDLA



Stripped Down MCL Application (MNIST)

```
/* handle command line arguments, declare variables, etc*/

char* dla_bin = "path to .nvdla binary"

float **in; //an array of digit images convert to 1-D arrays of floats
float **out; //an array of 10 element arrays specifying the predicted digit

/* input/output/ omitted for slides, please see full source for details */

mcl_handle** hdls =(mcl_handle**) malloc(num_inferences * size_of(mcl_handle*));
mcl_init(workers,0x0);

for (i=0;i<num_inferences;i++){
    hdls[i]=mcl_task_create();
    ret = mcl_task_set_kernel(hdls[i], dla_bin, "DLA_MNIST", 2, "", MCL_KERNEL_BIN));
    ret = mcl_task_set_arg(hdls[i], 0, (void*) in[i], sizeof(float)*IMGSIZE, MCL_ARG_INPUT | MCL_ARG_BUFFER));
    ret = mcl_task_set_arg(hdls[i], 1, (void*) out[i], sizeof(float)*10, MCL_ARG_OUTPUT | MCL_ARG_BUFFER));
    ret =  mcl_exec(hdls[h_idx], pes, NULL, MCL_TASK_NVDLA);
    // ret = mcl_wait(hdls[i]); //uncomment for synchronous execution
}
mcl_wait_all();

//print output predictions

for(i=0;i<num_inferences; i++){
    mcl_hdl_free(hdls[i]);
}
```

Currently compiled offline (previous slide)

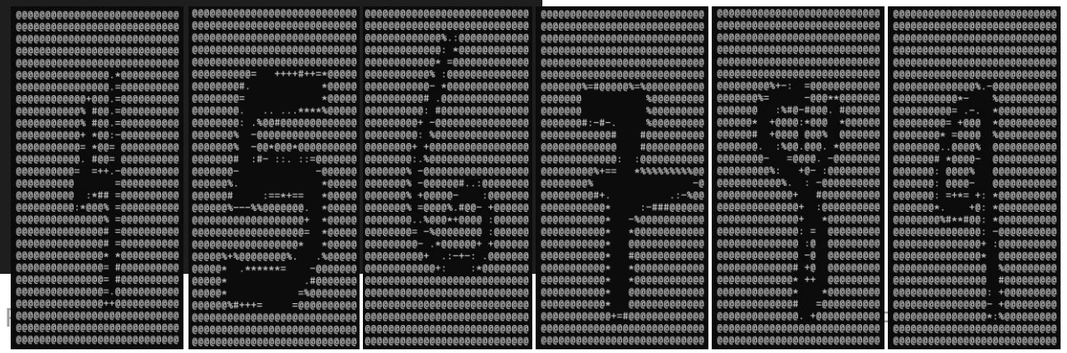
Pass DLA binary path when declaring kernel

Tell MCL this is a kernel binary

Force execution on an NVDLA

- We want to perform inference to predict what digit is present within an image
- The only thing we need to (potentially) change is the nvdla binary path to run on hardware vs emulated NVDLAs
- The docker container contains a full implementation of this application

```
root@container:/ppopp21/nvdla#
        make nvdla_test
```



Compiling NVDLA Binary and MCL Executable

```
# we want our qemu images running so open a new terminal → host:~/# docker exec -it ppopp21 /bin/bash
root@container:/ppopp21# cd nvdla/ → root@container:/ppopp21/nvdla# make caffe=mnist/mnist.caffemodel prototxt=mnist/mnist.prototxt nvdla_emu_bin
root@container:/ppopp21/nvdla# make nvdla_test
```

TERMINAL OUTPUT DEBUG CONSOLE 1: docker

```
-rwxrwxrwx    1 root     root        851933 May 21  2019 opendla_2.ko
-rwxr-xr-x    1 root     root        326472 May 21  2019 nvdla_runtime
-rwx------    1 root     root       8531103 May 21  2019 nvdla_compiler
-rwx------    1 root     root      57346646 May 21  2019 libnvdla_compiler.so
-rwxr-xr-x    1 root     root       3226960 May 21  2019 libnvdla_runtime.so
-rw-r-----    1 root     root      62914560 May 28  2019 rootfs.ext4
-rwxr-xr-x    1 root     root       1192864 Feb 25 23:37 nvdla_emu_server
drwxr-xr-x    1 root     root          4096 Feb 26 18:22 .
-rw-r-----    1 root     root      62914560 Feb 26 18:22 rootfs_6001.ext4
-rw-r-----    1 root     root      62914560 Feb 26 18:22 rootfs_6000.ext4
-rw-r--r--    1 root     root         28046 Feb 26 18:22 my_log_file.log
# LD_LIBRARY_PATH=. ./nvdla_emu_server
[    8.783665] opendla: loading out-of-tree module taints kernel.
[    8.807395] 0 . 12 . 5
[    8.808671] reset engine done
[    8.823591] [drm] Initialized nvdla 0.0.0 20171017 for 10200000.nvdla on minor 0
# pwd && ls -altr
/mnt
total 282265
drwxr-xr-x   18 root     root          1024 Nov 27  2017 ..
-rwx------    1 root     root           384 Nov 28  2017 init_dla.sh
-rw-r-----    1 root     root         64443 Dec 11  2017 LICENSE
-rw-r--r--    1 root     root        242176 Apr  6  2018 efi-virtio.rom
-rwxr-x---    1 root     root      16230912 Apr  6  2018 Image
-rw-r-----    1 root     root           733 Jul 11  2018 aarch64_nvdla_dump_dts.lua
-rw-r-----    1 root     root           708 Jul 11  2018 aarch64_nvdla.lua
-rwxrwxrwx    1 root     root      11363400 May 21  2019 drm.ko
-rwxrwxrwx    1 root     root        851677 May 21  2019 opendla_1.ko
-rwxrwxrwx    1 root     root        851933 May 21  2019 opendla_2.ko
-rwxr-xr-x    1 root     root        326472 May 21  2019 nvdla_runtime
-rwx------    1 root     root       8531103 May 21  2019 nvdla_compiler
-rwx------    1 root     root      57346646 May 21  2019 libnvdla_compiler.so
-rwxr-xr-x    1 root     root       3226960 May 21  2019 libnvdla_runtime.so
-rw-r-----    1 root     root      62914560 May 28  2019 rootfs.ext4
-rwxr-xr-x    1 root     root       1192864 Feb 25 23:37 nvdla_emu_server
drwxr-xr-x    1 root     root          4096 Feb 26 18:22 .
-rw-r-----    1 root     root      62914560 Feb 26 18:22 rootfs_6000.ext4
-rw-r-----    1 root     root      62914560 Feb 26 18:22 rootfs_6001.ext4
-rw-r--r--    1 root     root         30011 Feb 26 18:22 my_log_file.log
# LD_LIBRARY_PATH=. ./nvdla_emu_server
```

⊗0 ⚠0

Launching the MCL Scheduler

- Generally, just need to call `mcl_sched`
- Because we are using POCL + multiple emulated NVDLA we provide a wrapper script to help set some environment variables

`root@container:/ppopp21/nvdla#./launch_mcl_sched.sh`

```
#!/bin/bash

num_instances=`ls nvdla_emulator_configs/ | wc -l`
devices=""

for i in `seq 0 $(( num_instances-1 ))`; do
    devices="${devices}nvdlaemu " ← POCL device name
    port=$(( 6000+i ))
    export POCL_NVDLAEMU${i}_PARAMETERS="127.0.0.1:${port}"
done
POCL_DEVICES=${devices} mcl_sched &
```

List of devices for POCL to use

Running our MCL MNIST Inference application

- Similar to the MCL scheduler, we must set the same POCL environment variables, so we again wrap the nvdla_test binary in a script

```
root@container:/ppopp21/nvdla#./run_nvdla_test.sh
```

```
#!/bin/bash

num_instances=`ls nvdla_emulator_configs/ | wc -l`
devices=""

for i in `seq 0 $(( num_instances-1 ))`; do
        devices="${devices}nvdlaemu "
        port=$(( 6000+i ))
        export POCL_NVDLAEMU${i}_PARAMETERS="127.0.0.1:${port}"
done
POCL_DEVICES=${devices} ./nvdla_test
```

Lets See what Happens!

`root@container:/ppopp21/nvdla#./launch_mcl_sched.sh` → `root@container:/ppopp21/nvdla#./run_nvdla_test.sh`

PROBLEMS TERMINAL OUTPUT DEBUG CONSOLE 1: docker, docker

```
-rw-r-----    1 root     root      62914560 May 28  2019 rootfs.ext4
-rwxr-xr-x    1 root     root       1192864 Feb 25 23:37 nvdla_emu_server
drwxr-xr-x    1 root     root          4096 Feb 26 18:22 .
-rw-r-----    1 root     root      62914560 Feb 26 18:22 rootfs_6001.ext4
-rw-r-----    1 root     root      62914560 Feb 26 18:22 rootfs_6000.ext4
-rw-r--r--    1 root     root         28046 Feb 26 18:22 my_log_file.log
# LD_LIBRARY_PATH=. ./nvdla_emu_server
[    8.783665] opendla: loading out-of-tree module taints kernel.
[    8.807395] 0 . 12 . 5
[    8.808671] reset engine done
[    8.823591] [drm] Initialized nvdla 0.0.0 20171017 for 10200000.nvdla
on minor 0
# pwd && ls -altr
/mnt
total 282265
drwxr-xr-x   18 root     root          1024 Nov 27  2017 ..
-rwx------    1 root     root           384 Nov 28  2017 init_dla.sh
-rw-r-----    1 root     root         64443 Dec 11  2017 LICENSE
-rw-r--r--    1 root     root        242176 Apr  6  2018 efi-virtio.rom
-rwxr-x---    1 root     root      16230912 Apr  6  2018 Image
-rw-r-----    1 root     root           733 Jul 11  2018 aarch64_nvdla_du
mp_dts.lua
-rw-r-----    1 root     root           708 Jul 11  2018 aarch64_nvdla.lu
a
-rwxrwxrwx    1 root     root      11363400 May 21  2019 drm.ko
-rwxrwxrwx    1 root     root        851677 May 21  2019 opendla_1.ko
-rwxrwxrwx    1 root     root        851933 May 21  2019 opendla_2.ko
-rwxr-xr-x    1 root     root        326472 May 21  2019 nvdla_runtime
-rwx------    1 root     root       8531103 May 21  2019 nvdla_compiler
-rwx------    1 root     root      57346646 May 21  2019 libnvdla_compile
r.so
-rwxr-xr-x    1 root     root       3226960 May 21  2019 libnvdla_runtime
.so
-rw-r-----    1 root     root      62914560 May 28  2019 rootfs.ext4
-rwxr-xr-x    1 root     root       1192864 Feb 25 23:37 nvdla_emu_server
drwxr-xr-x    1 root     root          4096 Feb 26 18:22 .
-rw-r-----    1 root     root      62914560 Feb 26 18:22 rootfs_6000.ext4
-rw-r-----    1 root     root      62914560 Feb 26 18:22 rootfs_6001.ext4
-rw-r--r--    1 root     root         30011 Feb 26 18:22 my_log_file.log
# LD_LIBRARY_PATH=. ./nvdla_emu_server
```

```
ln: /usr/local/bin/gcc: File exists
ln: /usr/local/bin/g++: File exists
WE33900:~ frie869$ docker exec -it ppopp21 /bin/bash
root@3d648f75d945:/ppopp21# cd nvdla/
root@3d648f75d945:/ppopp21/nvdla# make caffe=mnist/mnist.caffemodel proto
txt=mnist/mnist.prototxt nvdla_emu_bin
LD_LIBRARY_PATH=/usr/local/nvdla /usr/local/nvdla/nvdla_compiler  --proto
txt mnist/mnist.prototxt --caffemodel mnist/mnist.caffemodel
creating new wisdom context...
opening wisdom context...
parsing caffe network...
libnvdla<3> mark prob
Marking total 1 outputs
attaching parsed network to the wisdom...
compiling profile "fast-math"... config "nv_full"...
closing wisdom context...
mv fast-math.nvdla mnist/mnist.nvdla
rm output.protobuf
rm -r wisdom.dir
root@3d648f75d945:/ppopp21/nvdla# ls mnist/
0.pgm  2.pgm  4.pgm  6.pgm  8.pgm  mnist.caffemodel  mnist.prototxt
1.pgm  3.pgm  5.pgm  7.pgm  9.pgm  mnist.nvdla
root@3d648f75d945:/ppopp21/nvdla# make CC=gcc-5 nvdla_test
gcc-5 -g -O3 nvdla.c -o nvdla_test -lmcl -lOpenCL -lm -lpthread -lrt
root@3d648f75d945:/ppopp21/nvdla# ls -altr
total 96
-rwxr-xr-x 1 root root  1261 Feb 25 23:54 start_nvdla_emulator.sh
-rwxr-xr-x 1 root root   292 Feb 25 23:54 run_nvdla_test.sh
-rw-r--r-- 1 root root   533 Feb 25 23:54 run_nvdla_emulator.exp
-rw-r--r-- 1 root root  8147 Feb 25 23:54 nvdla.c
-rwxr-xr-x 1 root root   270 Feb 25 23:54 launch_mcl_sched.sh
-rw-r--r-- 1 root root  4179 Feb 25 23:54 Readme.md
-rw-r--r-- 1 root root   897 Feb 25 23:54 Makefile
drwxr-xr-x 1 root root  4096 Feb 26 00:16 ..
drwxr-xr-x 2 root root  4096 Feb 26 18:22 nvdla_emulator_configs
drwxr-xr-x 1 root root  4096 Feb 26 18:40 mnist
drwxr-xr-x 1 root root  4096 Feb 26 18:41 .
-rwxr-xr-x 1 root root 36048 Feb 26 18:41 nvdla_test
root@3d648f75d945:/ppopp21/nvdla#
```

TL;DR (TL;DW)

```
host:~# docker run –name ppopp21 –it minoscomputing/ppopp21 /bin/bash
root@container:/ppopp21# cd nvdla/
root@container:/ppopp21/nvdla#./start_nvdla_emulator.sh 2
```

Start Emulated NVDLAs

```
# we want our qemu images running so open a new terminal
host:~/# docker exec -it ppopp21 /bin/bash
root@container:/ppopp21# cd nvdla/
root@container:/ppopp21/nvdla# make caffe=mnist/mnist.caffemodel prototxt=mnist/mnist.prototxt nvdla_emu_bin
root@container:/ppopp21/nvdla# make nvdla_test
root@container:/ppopp21/nvdla#./launch_mcl_sched.sh
root@container:/ppopp21/nvdla#./run_nvdla_test.sh
```

Compile NVDLA & MCL Binaries

Launch MCL Scheduler
Execute application

Pacific Northwest
NATIONAL LABORATORY

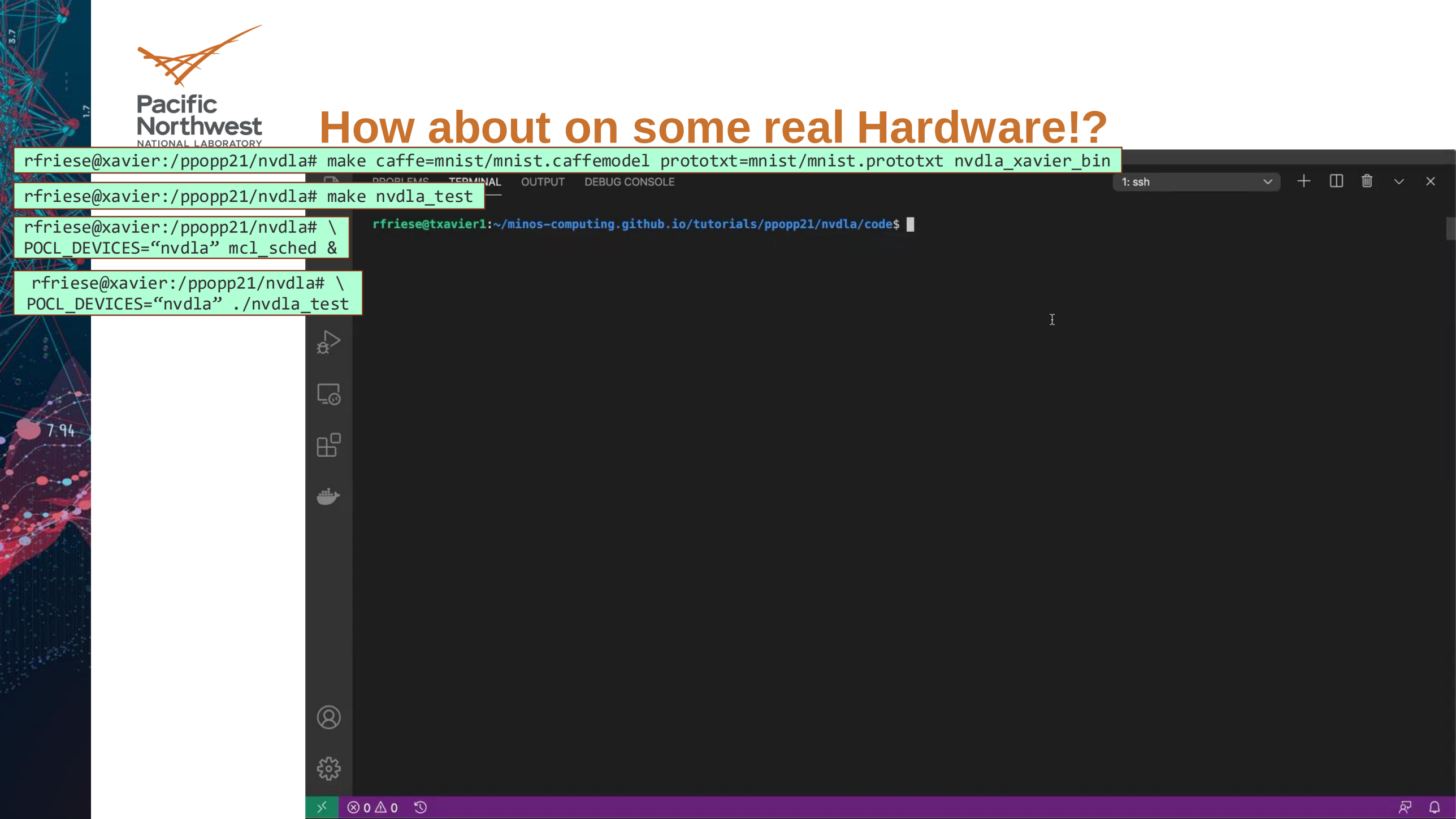
How about on some real Hardware!?

```
rfriese@xavier:/ppopp21/nvdla# make caffe=mnist/mnist.caffemodel prototxt=mnist/mnist.prototxt nvdla_xavier_bin
```

```
rfriese@xavier:/ppopp21/nvdla# make nvdla_test
```

```
rfriese@xavier:/ppopp21/nvdla# \
POCL_DEVICES="nvdla" mcl_sched &
```

```
rfriese@xavier:/ppopp21/nvdla# \
POCL_DEVICES="nvdla" ./nvdla_test
```



Pacific Northwest
NATIONAL LABORATORY

Thank you